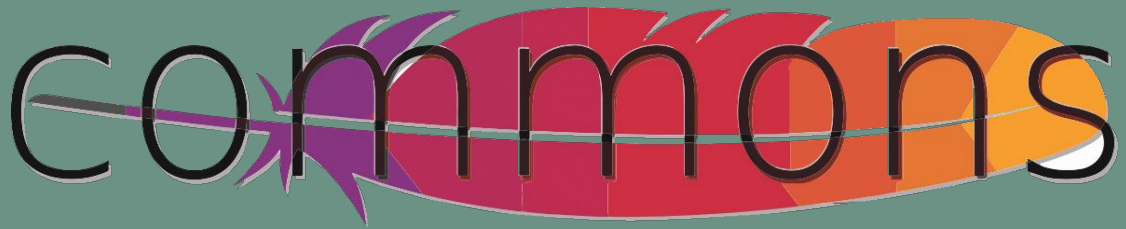
# Software Dependability Project Presentation

Daniele Fabiano and Francesco Maria Puca




Apache commons-net

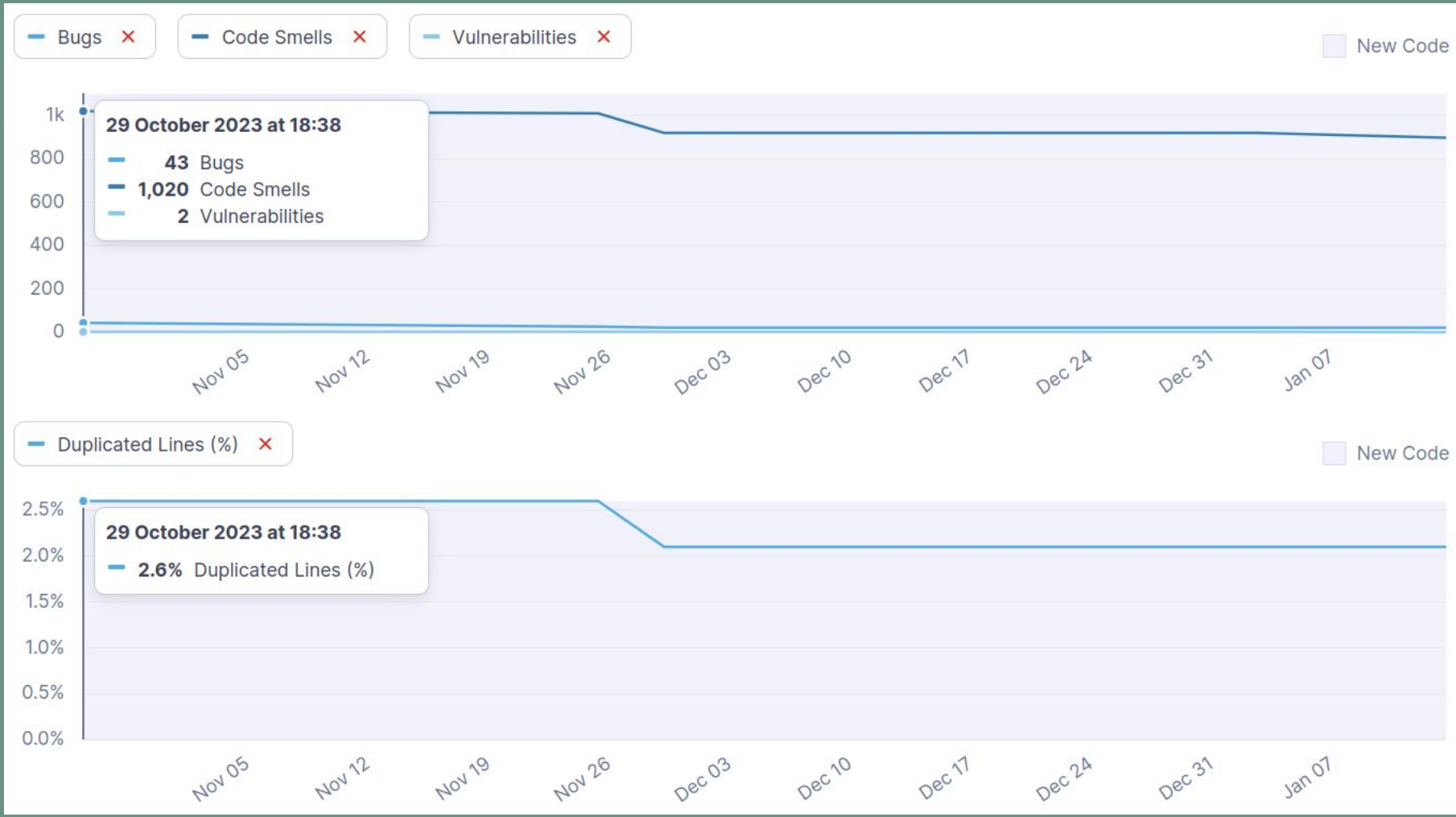
# What is Commons-net?

A popular Java library that implements the client side of many Internet protocols, such as FTP, NTP, POP3, SMTP and Telnet.

It is part of the Apache Commons Proper project, composed of reusable Java libraries with minimal dependencies.

Repository link: https://github.com/apache/commons-net

It has been active since 2002, with 33 contributors and 177 forks.


Bugs
Code Smells
Vulnerabilities
New Code
1k
800
600
400
200
0
29 October 2023 at 18:38
43 Bugs
1,020 Code Smells
2 Vulnerabilities
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dec 10
Dec 17
Dec 24
Dec 31
Jan 07
Duplicated Lines (%)
New Code
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
29 October 2023 at 18:38
2.6% Duplicated Lines (%)
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dec 10
Dec 17
Dec 24
Dec 31
Jan 07

# Goals for this project

- Fulfill all evaluation criteria

  (Buildability, SonarCloud, Coverage, Mutations, Performance, Automation, Security)

- Prioritize High Severity issues with Adaptability attribute

- Improve coverage reporting

- Inspect any software vulnerabilities present

# Bug-fixing and Refactoring - 1

19 bugs found with High Severity:

(2) Missing end conditions in loops: fixed by adding break conditions to while(true) blocks

(12) Try-with-resources without finally clause, fixed by declaring resources inside of existing try blocks to close them automatically.

# Bug-fixing and Refactoring - 2

(5) Try-with-resources without finally clause: Identified as False Positives

In these, resources declared inside of a try block were returned, so they should not be closed automatically.

These blocks haven’t been modified.

# Code Smells - 1

105 Code Smells found with High Severity, 72 of which were analyzed:

(6) Defining a constant instead of duplicating a literal:

Fixed by refactoring, using dedicated constants instead of literals.

(23) Adding at least one assertion to a test case: Identified as False Positives

Some of these test cases were inherited from interfaces, but not implemented.

Others passed the test automatically if no exceptions were thrown: therefore, they did not need any assertions.

# Code Smells - 2

(43) Refactoring a method to reduce its cognitive complexity:

In-depth knowledge of the codebase was required, as different coding styles were used, sometimes with complex control flows.

Most of these were "Brain Methods": high LOC and CC, fixed by dividing into multiple methods.

Simple when most variables involved were global, difficult when working with threads and exception handling.

Only one function was not refactored, due to a Cognitive Complexity of 150: too expensive in man-hours.

# Coverage Analysis - 1

JaCoCo results processed by Maven Plugin through Github Action and sent to CodeCov.

33.38% of computed coverage

5/19 packages with at least 50% of coverage:

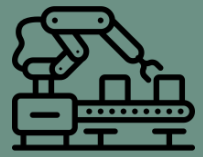
- ftp
- tftp
- util
- telnet
- ntp

# Coverage Analysis - 2

8/19 packages with 0% of coverage:

- discard
- whois
- echo
- pop3
- finger
- chargen
- daytime
- bsd

# Improving coverage - 1



Automatic creation of new test cases.

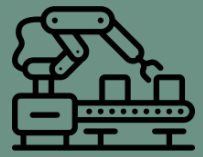
Different ways to generate test cases, but dependent on additional runtimes and heavy instrumentations.

Generated test cases not added to the original test suite and no computation of new coverage.

Notable improvement considering the possible impact for zero coverage packages.

We had nothing, now we have something.

# Improving coverage - 2



Evosuite execution: satisfying results, test cases generated for every package with zero coverage, at least half of the goals covered.

Randoop execution: unsuccessful execution for echo, chargen and daytime packages. Generation of flaky tests for discard package.

Methods from DatagramSocketClient, SocketClient and DiscardUDPClient are possible reasons for flakiness.

# Mutation Testing

Pitest was used, with its default configuration for mutant operators:

168 classes involved, resulting in:

37% Line coverage

30% Mutation coverage

78% Test strength.

# Performance Testing - 1

Java Microbenchmark Harness used to benchmark execution times during testing

Analyzed methods from the slowest test suites ( < 2 s/op):

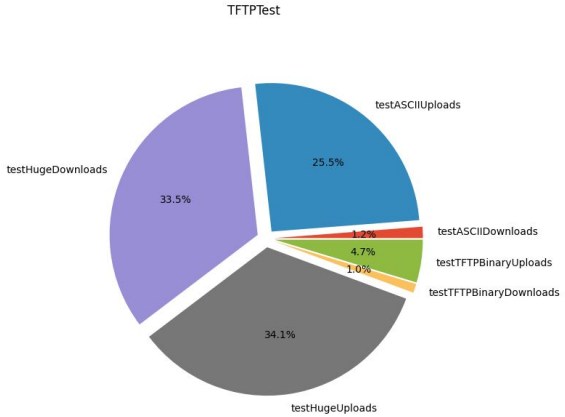
TestNtpClient - 2.009 s/op

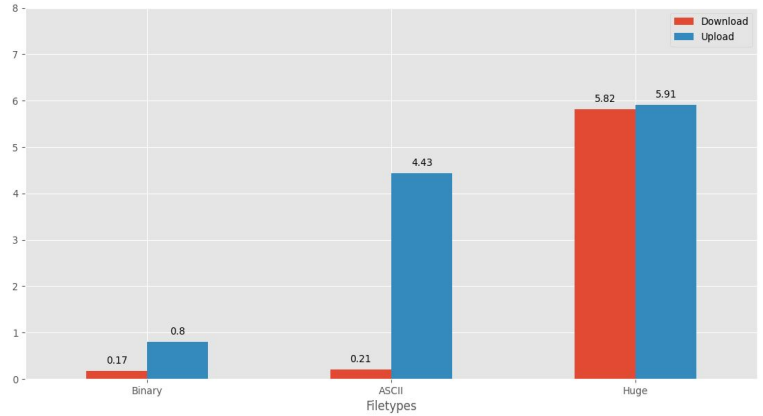
TFTPTest - 18.560 s/op

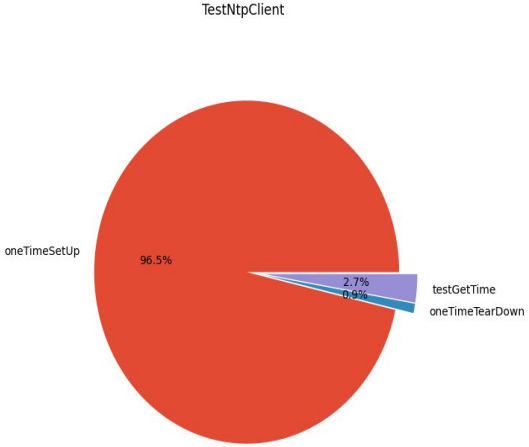
FTPSClientTest - 6.794 s/op

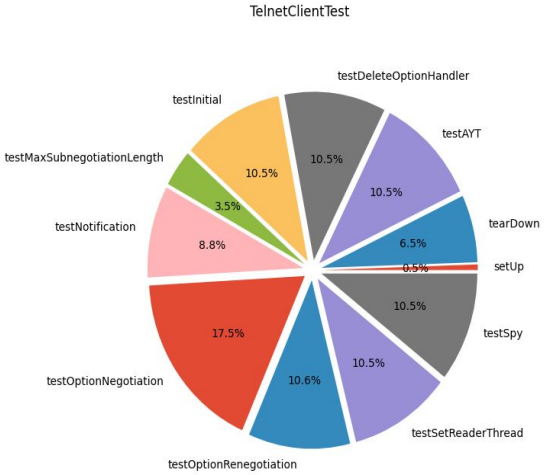
TelnetClientTest - 47.03 s/op

Annotations used: @State, @Benchmark


TFTPTest
testASCIIUploads
25.5%
testHugeDownloads
33.5%
1.2%
testASCIIDownloads
4.7%
testTFTPBinaryUploads
1.0%
testTFTPBinaryDownloads
34.1%
testHugeUploads


Download
Upload
8
7
6
5
4
3
2
1
0
5.82
5.91
4.43
0.8
0.17
0.21
Binary
ASCII
Huge
Filetypes


TestNtpClient
oneTimeSetUp
96.5%
2.7%
0.9%
testGetTime
oneTimeTearDown


TelnetClientTest
testDeleteOptionHandler
testInitial
testAYT
testMaxSubnegotiationLength
10.5%
10.5%
10.5%
3.5%
testNotification
tearDown
8.8%
6.5%
0.5%
setUp
10.5%
testSpy
17.5%
testOptionNegotiation
10.6%
10.5%
testSetReaderThread
testOptionRenegotiation

# Performance Testing - 2

FTPSClientTest - lacks public constructor, therefore incompatible with @State

TestNtpClient - oneTimeSetUp method was slowest due to Thread.sleep(1000), which was lowered to 100.

(2.009 s/op -> 0.111 s/op)

TFTPTest - sendFile method used a different type of InputStream, slower. However, it was not modified: needed to handle ASCII parsing.

TelnetClientTest - Thread.sleep() in all functions, reduced from 1000 to 100 when OutputStream was used and from 500 to 50 when InputStream was used.

(47.03 s/op -> 6.906 s/op)

# Software Vulnerabilities - 1

2 vulnerabilities found with High Severity:

(2) Server certificate validation on an SSL/TLS connection: first instance related to deprecated class and the second one identified as False Positive.

In the second case server-side certificates were correctly validated but erroneous notification to a lack of validation of client-side certificates

2 hotspots found with Medium Severity:

(1) Denial of Service: no security risk in the usage of the regex expression through pattern.compile()

(1) Weak Cryptography: no security risk in the usage of Random.nextInt() to generate a random number port

# Software Vulnerabilities - 2

FindSecBugs already executed by the original authors of the repository.

31 malicious code vulnerabilities found: Identified as False Positives

(14) May expose internal representation by returning reference to mutable object

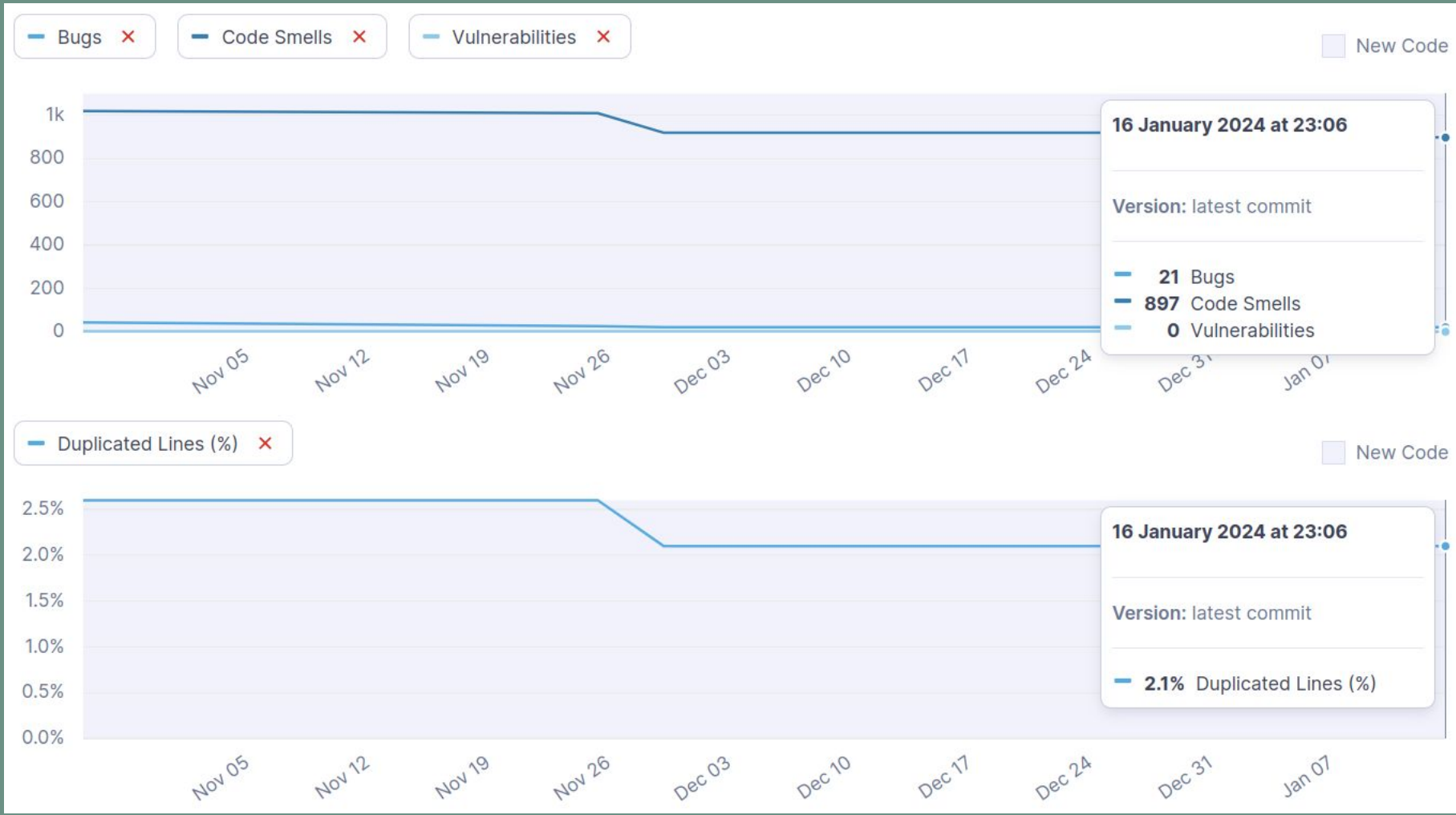
(16) May expose internal representation by incorporating reference to mutable object

(1) Public static method may expose internal representation by returning array: extra assessment to declare as False Positive, private and final arrays non accessible from the method

Dependency Check execution: no vulnerable dependencies found

# In conclusion:

| | |
|---|---|
| 22 | bugs fixed |
| 123 | code smells fixed |
| 2 | vulnerabilities fixed |
| 2 | security hotspots reviewed |
| 1793 | Evosuite goals reached |
| 33.4% | computed coverage |
| 45 s/op | saved |


Bugs
Code Smells
Vulnerabilities
New Code
1k
800
600
400
200
0
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dec 10
Dec 17
Dec 24
Dec 31
Jan 07
16 January 2024 at 23:06
Version: latest commit
21 Bugs
897 Code Smells
0 Vulnerabilities
Duplicated Lines (%)
New Code
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dec 10
Dec 17
Dec 24
Dec 31
Jan 07
16 January 2024 at 23:06
Version: latest commit
2.1% Duplicated Lines (%)

# That's it!

Any Questions?